NATIONAL UNIVERSITY OF SINGAPORE

CS1010 – PROGRAMMING METHODOLOGY

(Semester 2: AY2016/17)

Time allowed: 2 hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains **SEVEN** questions and comprises **EIGHT** printed pages, including this page.
2. This is an **OPEN BOOK** assessment. The maximum possible score is **55 marks**.
3. Calculators are allowed, but not laptops or other electronic devices.
4. Write all your answers in the **ANSWER SHEETS** provided.
5. Submit only the **ANSWER SHEETS** at the end of the assessment. You may keep the question paper.
6. Do **NOT** look at the questions until you are told to do so.

Q1. **[Total: 12 marks]**

1.1 Write down the output of the following program fragment. [1 mark]

```
char str[] = "abcde";
str[3] = 0; // assign integer zero
puts(str);
```

1.2 Write down the return value of the function call `foo(123)`. [2 marks]

```
int foo(int num) {
  if (num < 10) {
    return num*num;
  } else {
    return foo(num/10) + (num%10)*(num%10);
  }
}
```

1.3 Write down the output of the following program. [3 marks]

```
#include <stdio.h>

int f(int *p, int q);

int main(void) {

  int x = 2, y = 5, z;

  z = f(&x, y);

  printf("%d %d %d\n", x, y, z);

  return 0;
}

int f(int *p, int q) {
  (*p)++;
  q--;
  return 2*q + (*p);
}
```

1.4 Write down the output of the following program. [3 marks]

```
#include <stdio.h>

int main(void) {

  int x = 100, i = 0, j;

  do {
    for (j = 0; j < 20; j++) {
      if ( !(j%2) ) {
        x += j;
      } else {
        x += j-1;
      }
    }
    i++;
  } while (i < 10);

  printf("%d\n", x);

  return 0;
}
```

1.5 Write down the output of the following program. [4 marks]

```
#include <stdio.h>
#define SIZE 4

int check(int mtx[SIZE][SIZE]);

int main(void) {

  int a[SIZE][SIZE] = {{0, 0, 0, 9},
                       {0, 0, 1},
                       {0, 8, 0, 1}};

  int b[SIZE][SIZE] = {{0}, {0}, {0}, {3}};

  if (check(a)) {
    printf("True ");
  } else {
    printf("False ");
  }

  if (check(b)) {
    printf("True\n");
  } else {
    printf("False\n");
  }

  return 0;
}

int check(int mtx[SIZE][SIZE]) {

  int row, col;

  for (row = 0; row < SIZE; row++) {
    for (col = 0; col < SIZE; col++) {
      if (row+col != SIZE-1 && mtx[row][col] != 0) {
        return 0;
      }
    }
  }

  return 1;
}
```

Q2. **[Total: 13 marks]**

A supermarket sells different types of items, each of which is identified by a **unique** item identification number.

To describe an item, an `item_t` structure is defined as follows:

```
typedef struct {
    int id;          // identification number of an item
    double price;    // price of an item
} item_t;
```

(a) **[4 marks]** Write a function `min_max_price()` that takes an array of `item_t` variables, returns the prices of the cheapest and most expensive items through two pointer parameters.

Function prototype is as follows.

```
void get_min_max_price(item_t items[], int size,
                       double *min_p, double *max_p);
```

Parameter `size` is the number of elements in `items` array.

(b) **[4 marks]** Write a function `search_item()` that takes an array of `item_t` variables and an identification number `id`, checks if any item in the array has such an identification number. This function returns the index of such an item in the array if found, or returns -1 otherwise.

Function prototype is as follows.

```
int search_item(item_t items[], int size, int id);
```

Parameter `size` is the number of elements in `items` array. You may assume that the `items` array is already sorted in ascending order of identification numbers of items.

(c) **[5 marks]** Write a function `sort_items()` that takes an array of `item_t` variables, sorts the items in decreasing order of prices, and for those items with the same price, in increasing order of their identification numbers.

Function prototype is as follows.

```
void sort_items(item_t items[], int size);
```

Parameter `size` is the number of elements in `items` array.

Q3. **[Total: 10 marks]**

Run-length encoding (RLE) is a very simple form of data compression. Its idea is illustrated with the following example.

Suppose we have an array of integers like this:

```
3 3 3 2 2 2 2 7 7 7 2 2 2 6 9 9 9 9 5 27 27 27 42
```

We will replace each integer with a **count** of the number of times that integer occurs consecutively, followed by that **integer**. So the above list of integers may be compressed to a shorter list:

```
3 3 4 2 3 7 3 2 1 6 4 9 1 5 3 27 1 42
```

(a) **[5 marks]** Write a function **`compress()`** that compresses data using RLE. Function prototype is as follows.

```
int compress(int arr[], int size, int comp[])
```

The first parameter **`arr`** is the array of integers to be compressed; the second parameter **`size`** is the number of integers in **`arr`**. The compressed data is written into the array **`comp`**. You may assume **`comp`** array has sufficient space to accommodate the compressed list.

This function returns the number of integers in **`comp`**, inclusive of the counts for each integer. In the above example, this function will return 18.

(b) **[5 marks]** Write a function **`decompress()`** to decompress a stream of integers compressed with RLE.

For example, given the following compressed list of integers:

```
1 3 5 26 7 9 3 15
```

This function will produce

```
3 26 26 26 26 26 9 9 9 9 9 9 9 15 15 15
```

Function prototype is as follows.

```
int decompress(int comp[], int size, int decomp[])
```

The first parameter **`comp`** is the array of integers to be decompressed; the second parameter **`size`** is the number of integers in **`comp`**. The decompressed data is written into the array **`decomp`**. You may assume **`decomp`** array has sufficient space to accommodate the decompressed list.

This function returns the number of integers in the decompressed array. In the above example, this function will return 16.

Q4. **[4 marks]**

Re-write the following function **what()** without using any selection/repetition statement. You may define additional variables and use functions from <math.h> as necessary.

```
double what(double num1, double num2, double num3) {
  if ( (num1 >= num2 && num1 <= num3) ||
       (num1 >= num3 && num1 <= num2) ) {
    return num1;
  } else if ( (num2 >= num1 && num2 <= num3) ||
       (num2 >= num3 && num2 <= num1) ) {
    return num2;
  } else {
    return num3;
  }
}
```

Q5. **[5 marks]**

Write a recursive function

```
int all_odd(int num)
```

that takes a positive integer **num** as parameter and returns 1 if all digits of **num** are odd, or 0 otherwise. Note that 0 is an even number.

For example,

- Function call **all_odd(5)** should return 1;
- Function call **all_odd(157)** should return 1;
- Function call **all_dd(990)** should return 0;
- Function call **all_odd(4)** should return 0.

You are **NOT** allowed to use any loop constructs (*for*, *while* or *do-while*) in this question.

Q6. **[5 marks]**

Write a function

```
int sum_integers(char str[])
```

that returns the sum of all the integers in the string **str**. You may assume that **str** contains alphabet and/or digits only.

For example,

- Function call on string "**b123def4g56**" should return 183 (= 123 + 4 + 56);
- Function call on string "**b123**" should return 123;
- Function call on string "**abc**" should return 0;
- Function call on string "**7**" should return 7.

You may feel free to use library functions from <ctype.h> and <string.h>.

Q7. **[5 marks]**

Write a function

```
void fill_array(int mtx[N][N])
```

that takes a two-dimensional array **mtx** of **N** rows and **N** columns as parameter, and fills it with integers from 0 to **N***(**N** -1) as follows:

(1) The main diagonal should be filled with 0s.
(2) Odd integers (starting from 1) should appear below the main diagonal. They will be filled in along diagonals, starting from the lowest diagonal.
(3) Even integers (starting from 2) should appear above the main diagonal. They will be filled in along diagonals, starting from the highest diagonal.

You may assume that (1) **mtx** is uninitialized when the function call **fill_array(mtx)** is made; (2) **N** is a constant defined in the program whose value is a positive integer greater than 2. Two examples are given below.

Odd integers are filled in along diagonals. So do evens.

0	8	4	2
7	0	10	6
3	9	0	12
1	5	11	0

N = 4

0	14	8	4	2
13	0	16	10	6
7	15	0	18	12
3	9	17	0	20
1	5	11	19	0

N = 5

=== END OF PAPER ===